Randal L. Schwartz
Stonehenge Consulting Services, Inc.
+1 (503) 777 0095
http://www.stonehenge.com/merlyn/

Introduction to CGI using CGI.pm

Randal L. Schwartz
Stonehenge Consulting Services, Inc.
www.stonehenge.com/merlyn/
version 1.4, 25 Feb 2005

What we're covering

- *Review of basic web processing*
- *CGI.pm basic syntax and usage*
- *HTML generation*
- *Form handling*
- *All the details*

Basic Web Processing

- *Browser asks for a URL*
- *Server returns a page*
- *If the page is HTML, browser displays it with links and images*
- *If a link is selected, the process repeats*
- *The server is not involved between hits*

Forms Processing

- *Browser asks for a URL*
- *Server returns a page containing an HTML form*
- *User fills out the form, and "submits"*
- *Browser posts form data to hidden URL*
- *Server examines form data, and creates appropriate response*

Example

- *Browser asks for a URL:*

```
GET /somedir/page.html
```

- *Server replies with form:*

```
<html>...<form action='response.cgi'>
<input type=text name=first_name>
<input type=text name=last_name>
<input type=submit>
</form>...</html>
```

Example (continued)

- *User fills in "Fred" "Flintstone" and hits submit*
- *Browser sends:*

```
POST /somedir/response.cgi
first_name=Fred
last_name=Flintstone
submit=submit
```

Example (continued)

- *Server responds with HTML page from the CGI script:*

```
<html>...
Hello, Fred Flintstone!
...</html>
```

Responding with CGI.pm

- *response.cgi might look like:*

```
use CGI qw(param);
print “Content-type: text/html\n\n”;
my $first = param(”first_name”);
my $last = param(”last_name”);
print “<html><head><title>Hello!</title></head>”;
print “<body><h1>Hi!</h1>”;
print “Hello, $first $last!”;
print “</body></html>”;
```

- *Note that the CGI script is also responsible for a CGI header*

Eliminating all those <>'s

- *Using HTML shortcuts:*

```
use CGI qw(:all);
my $first = param('first_name');
my $last = param('last_name');
print header, start_html('Hello');
print h1('Hi!');
print "Hello, $first $last!";
print end_html;
```

- *Much shorter, and easier to read*

Using form-generation shortcuts

- *Printing the original form:*

```
use CGI qw(:all);
print header, start_html('Hello');
print h1('Your name, please');
print start_form;
print textfield('first_name'), textfield('last_name');
print submit, end_form;
print end_html;
```

- *Also much shorter, and easier to read*

Putting them in one script

- *Either the form or the response:*

```
use CGI qw(:all);
print header, start_html('Hello');
if (param) { # I've seen some params
  print h1('Hello!'), 'Hello, ',
    param('first_name'), ' ', param('last_name'), '!';
} else {
  print h1('Your name, please');
  print start_form,
    textfield('first_name'),textfield('last_name'),
    submit, end_form;
}
print end_html;
```

From start to finish

- *header—prints the content type:*
  `header($content_type)`
- *redirect—send them somewhere else (use this in place of header):*
  `redirect($new_url)`
- *start_html—prints the title:*
  `start_html('This page left blank')`
- *end_html—ends the HTML*

HTML formatting

- *h1: first-level header:*
  ```
  h1('This is a sample page')
  ```
- *Similarly, h2 through h6*
- *In fact, all the HTML tags*
- *Attributes indicated with hashrefs:*
  ```
  print a({-href = "http://www.google.com"}, "Google");
  ```
- *Attributes automatically HTML escaped*
- *Text isn't, use escapeHTML:*
  ```
  print b(escapeHTML($unknown_string))
  ```

Getting the parameters

- *param—fetch the parameter:*

```
my $text = param('this');
```

- *Also used for multiple parameters:*

```
my @selected = param('select_state');
```

- *Update the parameter (for sticky fields):*

```
param('this', 'new_value');
param('select_state', qw(Oregon Idaho));
```

Generating forms

- *start_form($method,$action)*
- *Defaults to POST and current page*
- *end_form*
- *The form consists of various input types, such as textfield and submit*

A slight aside: the CGI.pm arg-er

- *CGI.pm has positional or named arguments:*

```
start_form('POST', '/cgi/foo');
start_form(-method => 'POST', -action => '/cgi/foo');
start_form(-action => '/cgi/foo');
```

- *Positional args take defaults for unspecified later args*
- *Named args take defaults for all unmentioned args*

Textfields

- *textfield($name, $default, $size, $maxlength)*
- *$name is fieldname*
- *$default is default value (if no sticky value has been supplied)*
- *$size is screen width*
- *$maxlength puts a browser limit on maximum length*

Sticky fields

- *The default value for a form element comes from the incoming parameter of the same name*
- *If no incoming parameter is supplied, the $default value is used*
- *If neither is provided, no default is presented to the browser*

Managing sticky fields

- *Usually, sticky fields do the right thing*
- *When you re-present a form to be re-entered for an error, it already has the previous values, nice for the user*
- *Override sticky fields using param()*
- *Delete a sticky field with Delete():*

```
Delete(qw(first_name last_name));
delete_all(); # ignore sticky
```

Textareas

- *textarea($name, $default, $rows, $columns)*
- *$name is fieldname*
- *$default is default value*
- *$rows is screen height*
- *$cols is screen width*

Pop-up Menus

- *popup_menu($name, $values, $default, $labels)*
- *$name is field name*
- *$values is arrayref of values*
- *$default selects one of the values*
- *$labels is hashref mapping @$values to the human-readable labels*

Scrolling list

- *scrolling_list($name, $values, $default, $size, $multiple, $labels)*
- *$name, $values, $default, $labels like popup_menu*
- *$size is visible size*
- *$multiple is true for multiple select*
- *Use param() in list context for multiple*

Checkbox Group

- *checkbox_group($name, $values, $default, $linebreak, $labels)*
- *$name, $values, $default, $labels like popup_menu*
- *If $linebreak is true, make it vertical*
- *Use param() in list context, or you'll end up seeing a random "first" item*

Radio Group

- *radio_group($name, $values, $default, $linebreak, $labels)*
- *Just like checkbox_group, but with radio buttons*
- *No need for param() in list context: only one value will ever be selected*

Checkbox

- *checkbox($name, $on, $value, $label)*
- *If $on is true, it's on*
- *But the sticky value will override the given default*

Submit Button

- *submit($name, $value)*
- *$name is what the user sees*
- *$value is what gets passed back as the value for param $name (defaults to $name)*
- *$name defaults to "Submit" and sends "Submit=Submit" when pressed*
- *Always include a submit button*

Image button

- *image_button($name, $src)*
- *$src is the URL to the image*
- *Returns as params of "$name.x" and "$name.y"*

Hidden fields

- *hidden($name, $default)*
- *Useful for passing state information or session tracking*
- *$default can be an array ref, and extracted with list-context param()*
- *Don't trust hidden data*
- *"view source" shows "hidden"*

Fetching Cookies

- *$value = cookie($name)*
- *Fetches the current value*
- *List context retrieves multiple values*

Creating a Cookie

- *cookie($name, $value, $path, $domain, $secure, $expires)*
- *$path, $domain, and $secure control when the cookie is sent*
- *$expires permits flexible specifications:*
  - *+30s: 30 seconds from now*
  - *+3M: 3 months from now*
  - *now*

Setting the Cookie

- *Include one or more cookies in the header:*

```
my $c1 = cookie('this', 'value');
my $c2 = cookie('that', 'value');
print header(-cookie => [$c1, $c2]);
```

- *Remember that cookies sent in this header are not visible until the next hit*
- *Also, a user may refuse a cookie, manually or automatically*

Self-referencing URLs

- *self_url: reload current page with all params repeated*
- *url($option1 => 1, $option2 => 1, ...)*
  - *-absolute: starts with slash*
  - *-relative: starts with script name*
  - *-full: starts with http:...*
  - *-path: retain pathinfo information*
  - *-query: retain query string*

Object-oriented interface

- *Instead of "use CGI qw(:all)":*

```
use CGI;
my $q = CGI->new;
print $q->header, $q->start_html, $q->h1('hello');
...
```

- *A little silly for something that's only one*
- *Requires a lot more typing*
- *Frees up all those imports and collisions*
- *Also handy when used with TT*

For more information

- *perldoc "CGI"*
- *Ovid's CGI tutorial at:*
  `http://users.easystreet.com/ovid/cgi_course/`